CONTENTS

6 Model Fitting

6.1 Linear Regression With Python

6.1.1 Simple and Multiple Regression

Step 1: Import packages

First you need to do some imports. In addition to numpy, you need to import statsmodels.api:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Now you have the packages you need.

Step 2: Provide data and transform inputs

The second step is defining data to work with. The regressors,x and predictor, y should be arrays.

```
y = np.array([5, 20, 14, 32, 22, 38])
n = len(y)
x0 = np.ones(n).reshape((-1, 1))
```

```
x1 = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
x = np.hstack((x0, x1))
```

Now, you have two arrays: the x and y. You should call .reshape() on x because this array is required to be two-dimensional, or to be more precise, to have one column and as many rows as necessary. That's exactly what the argument (-1, 1) of .reshape() specifies.

Step 3: Create a model and fit it

The regression model based on ordinary least squares is an instance of the class

statsmodels.regression.linear_model.OLS.

This is how you can obtain one:

```
model = sm.OLS(y, x)
```

Once your model is created, you can apply .fit() on it:

```
results = model.fit()
```

By calling .fit(), you obtain the variable results, which is an instance of the class statsmodels.regression.

linear_model.RegressionResultsWrapper.

This object holds a lot of information about the regression model.

Step 4: Get results

You can call .summary() to get the table with the results of linear regression:

```
print(results.summary())
```

```
 OLS Regression Results
==============================================================================
Dep. Variable:            y   R-squared:                       0.716
Model:                  OLS   Adj. R-squared:                  0.645
Method:      Least Squares   F-statistic:                     10.08
Date:       Fri, 29 Oct 2021   Prob (F-statistic):             0.0337
Time:              17:19:23   Log-Likelihood:                -19.071
No. Observations:        6   AIC:                             42.14
Df Residuals:            4   BIC:                             41.73
Df Model:                1
Covariance Type:            nonrobust
==============================================================================
```

```
        coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const  5.6333      5.872      0.959      0.392     -10.670      21.936
x1     0.5400      0.170      3.175      0.034       0.068       1.012
==============================================================================
Omnibus:                      nan   Durbin-Watson:                   3.606
Prob(Omnibus):                nan   Jarque-Bera (JB):                0.651
Skew:                       0.008   Prob(JB):                        0.722
Kurtosis:                   1.387   Cond. No.                         69.8
==============================================================================
```

Step 5: Predict response

You can obtain the predicted response on the input values used for creating the model using .fittedvalues or .predict() with the input array as the argument:

```
print('predicted response:',results.fittedvalues)
print('predicted response:',results.predict(x))
```

```
predicted response: [ 8.33333333 13.73333333 19.13333333 24.53333333
                      29.93333333 35.33333333]
```

This is the predicted response for known inputs. If you want predictions with new regressors, you can also apply .predict() with new data as the argument:

```
x_new = np.array([1, 20])
y_new = results.predict(x_new)
print("y_new=",y_new)
```

6.1.2 Testing for significance of regression

Is the regression equation that uses information provided by the predictor variables $x_1, x_2, \ldots, x_k$ substantially better than the simple predictor y that does not rely on any of the $x-$values? This question is answered using an overall F test.

1. The hypotheses are $H_0 : \beta_1 = \cdots = \beta_k = 0$ versus H_1 : At least on β's $\neq 0$
2. The test statistic is F
3. Find the p-value
4. Reject H_0 if p-value $< \alpha$, where α is the level of significance, usually is 5%.

When the null hypothesis is rejected, this means that that the model using $x_1, x_2, \cdots, x_k$ as predictor variables is useful for y.
We can obtain the test statistic, F, and the p-value from the summary. If we want to obtain the test statistic, F, and the p-value seperately, we can use the following.

```
F = results.fvalue
print(‘‘F =’’,F)
pv = results.f_pvalue
print{‘‘P-value = ‘‘, pv)
```

6.1.3 Testing the Significance of the Partial Regression Coefficients

Once you have determined that the model is useful for predicting y, you should explore the nature of the "usefulness" in more detail. Do all the predictor variables add important information for prediction in the presence of other predictors already in the model?

1. The hypotheses are $H_0 : \beta_j = 0$ versus $H_1 : \beta_j \neq 0$ for $j = 1, 2, \ldots, k$
2. The test statistics is $t = \dfrac{\hat{\beta}_j}{SE(\hat{\beta}_j)}$, where $SE(\hat{\beta}_j)$ is the standard error of $\hat{\beta}_j$
3. Find the p value
4. Reject H_0 if p-value $< \alpha$.

Similarly we can obtain the test statistic, t, and the p-value from the summary. If we want to obtain the test statistic, t, and the p-value seperately, we can use the following.

```
t = results.tvalues[j]
print(‘‘t =’’,t)
pv = results.pvalues[j]
print{‘‘P-value = ‘‘, pv)
```

6.1.4 Regression Performance

In topic 5 , we know that the larger the R^2, the better fit the data to the model. However, R^2 increases as the number parameters increases. Thus we usually will use adjusted R^2 to choose the appropriate model. Another two quantity that use to make decision on model selection are Aikaike Information Criterion(AIC) and Bayesisn Information Criterion(BIC). Smallest AIC or BIC indicate the best model. R^2, adjusted R^2, AIC ans BIC can be obtained from the summary as well. To get them seperately, we use the following.

```
Rsq = results.rsquared
print(‘‘Rsq = ‘‘, Rsq)
Adj_Rsq = results.rsquared_adj
print(‘‘Adj_Rsq = ‘‘, Adj_Rsq)
AIC = results.aic
Print(‘‘AIC = ’, AIC)
BIC = results.bic
Print(‘‘BIC = ’, BIC)
```

Example 1.

An individual claims that the fuel consumption of his automobile does not depend on how fast the car is driven. To test the plausibility of this hypothesis, the car was tested at various speeds between 70 and 120 kilometers per hour. The kilometer per liter attained at each of these speeds were determined, with the following data resulting.

Speed	Kilometer per liter
70	10.29
80	10.63
90	99.06
100	93.53
110	89.28
120	80.78

You fit the above data to $Y = \beta_0 + \beta_1 X + \epsilon$, where Y is the kilometer per liter, and X is the speed of the car. Use the stat.models module to answer the following questions:

(a) State the estmated regression function.

(b) Obtain a prediction for a new observation y_h when $x_h = 98.0$.

(c) Compute the R^2.

(d) Compute the adjusted R^2.

(e) Compute the AIC.

(f) Compute the BIC.

(g) Compute the test statistic for testing $H_0 : \beta_1 = 0$ versus $H_1 : \beta_1 \neq 0$.

(h) Compute the p value corresponding to the test above.

Sol:

(a) $\hat{Y} = -94.27923809523831 + 1.6653428571428595X$

```
import numpy as np
import statsmodels.api as sm
y = ([10.29,10.63,99.06,93.53,89.28,80.78])
n = len(y)
x0 = np.ones(n).reshape(-1,1)
x1 = np.array([70, 80, 90, 100, 110, 120]).reshape(-1,1)
x = np.hstack((x0, x1))
x,y = np.array(x), np.array(y)
model = sm.OLS(y, x)
results = model.fit()
print(results.summary())
b = results.params
print("b = ", b)
Output:
b =  [-94.27923809523831 1.6653428571428595]
```

(b) $y_h = -94.2792 + 1.6653(98.0) = 68.9244$

```
Python code:
xh = np.array([1,98.0])
yh = results.predict(xh)
```

```
print("yh = ",yh)
Output:
yh = [[68.9243619]]
```

(c) $R^2 = 0.554359030014413$

```
print("R-squared = ", results.rsquared)
Output:
R-squared =  0.554359030014413
```

(d) Adjusted $R^2 = 0.44294878751801625$

```
print("Adj. R-squared = ", results.rsquared_adj)
Output:
Adj. R-squared =  0.44294878751801625
```

(e) AIC $= 59.8915107479189$

```
print("AIC = ", results.aic)
Output:
AIC =  59.8915107479189
```

(f) BIC $= 59.47502968637501$

```
print("BIC = ", results.bic)
Output:
BIC =  59.47502968637501
```

(g) $t = 2.230658067313043$

```
print("t = ", results.tvalues[1])
Output:
t =  2.230658067313043
```

(h) $pvalue = 0.08954545423334709$

```
print("p value = ", results.pvalues[1])
Output:
p value =  0.08954545423334709
```

Example 2.

The data file VegiB.csv record a flavor and texture score, Y (between 0 and 100) for 12 brands of meatless hamburgers along with the price, X_1, number of calories, X_2, amount of fat, X_3, and amount of sodium per burger, X_4. Assuming that regression model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$ is appropriate.

(a) Fit a regression model to the data. State the estimated regression function.

(b) Determine the predicted value for the mean score of flavor and texture for all brands of meatless hamburgers with the price = 83, number of calories = 141, amount of fat = 5, and amount of sodium per burger = 487.

(c) Is the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \epsilon$ useful for estimating the texture score? The hypotheses are $H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$ versus $H_1 : \beta_1 = \beta_2 = \beta_3 = \beta_4 \neq 0$. Find the test statistic for the hypotheses.

(d) Find the corresponding p value for testing $H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$.

Sol:

(a) $\hat{Y} = 59.8488 + 0.1287X_1 + -0.5805X_2 + 8.4982X_3 + 0.0488X_4$

Python code:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
df = pd.read_csv('VegieB.csv')
y = df.y
n = len(y)
x0 = np.ones(n).reshape(-1,1)
x1 = np.array(df.x1).reshape(-1,1)
x2 = np.array(df.x2).reshape(-1,1)
x3 = np.array(df.x3).reshape(-1,1)
x4 = np.array(df.x4).reshape(-1,1)
x = np.hstack((x0,x1,x2,x3,x4))
x, y = np.array(x), np.array(y)
model = sm.OLS(y, x)
results = model.fit()
b = results.params
print("beta = ",b)
Output:
beta =  [59.8488 0.1287 -0.5805 8.4982  0.0488 ]
```

(b) $\hat{Y}_h = 59.8488 + 0.1287(83) + -0.5805(141) + 8.4982(5) + 0.0488(487) = 54.92021064858727$

```
Python code:
x_new = np.array([1,83,141,5,487]).reshape(-1,5)
print("x_new=",x_new)
y_new = results.predict(x_new)
print("y_new=",y_new)
Output:
beta =  [59.8488 0.1287 -0.5805 8.4982  0.0488 ]
x_new= [[1  83  141 5   487]]
y_new= [54.92021064858727]
```

(c) $F = 1.7436100388886453$

```
Python code:
```

```
F = results.fvalue
print("F = ",F)
Output:
F = 1.7436100388886453
```

(d) p value = 0.24434282165926988

```
Python code:
pv = results.f_pvalue
print("p value = ",pv)
Output:
p value = 0.24434282165926988
```

6.1.5 Polynomial Regression

Implementing polynomial regression is very similar to linear regression. There is only one extra step: you need to transform the array of inputs to include non-linear terms such as x^2.

Step 1 Import packages and classes

In addition to numpy and statsmodels, you should also import the class

PolynomialFeatures from sklearn.preprocessing:

```
import numpy as np
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
```

Step 2 Provide data and transform input data

The input and output and is the same as in the case of linear regression:

```
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([15, 11, 2, 8, 25, 32])
```

As you've seen earlier, you need to include x^2 (and perhaps other terms) as additional features when

implementing polynomial regression. For that reason, you should transform the input array x to contain the additional column(s) with the values of x^2 (and eventually more features). It's possible to transform the input array in several ways (like using insert() from numpy), but the class PolynomialFeatures is very convenient for this purpose. Let's create an instance of this class:

```
transformer = PolynomialFeatures(degree=2,
                                 include_bias=False)
```

The variable transformer refers to an instance of PolynomialFeatures which you can use to transform the input x.

You can provide several optional parameters to PolynomialFeatures:

- degree is an integer (2 by default) that represents the degree of the polynomial regression function.
- interaction_only is a Boolean (False by default) that decides whether to include only interaction features (True) or all features (False).
- include_bias is a Boolean (True by default) that decides whether to include the bias (intercept) column of ones (True) or not (False).

Before applying transformer, you need to fit it with .fit():

```
transformer.fit(x)
```

Once transformer is fitted, it's ready to create a new, modified input. You apply .transform() to do that:

```
x_ = transformer.transform(x)
```

You can also use .fit_transform() to replace the three previous statements with only one:

```
x_ = PolynomialFeatures(degree=2,
                        include_bias=False).fit_transform(x)
```

That's fitting and transforming the input array in one statement with .fit_transform().

Example 3.

Consider the data shown below:

y	x	y	x
1,191,032	32	445,462	25
183,297	20	267,807	22
183,297	20	1,049,467	31
96,071	17	24,124	12
699,577	28	222,556	21
11,726	10	222,556	21
1,346,458	33	804,549	29
149,477	19	96,071	17

Fit the polynomial regression models up to order 4, then answer the following questions:

(a) Fill in the table below:

Polynomial model	Adjusted R^2	AIC	BIC
First order			
Second order			
Third order			
Fourth order			

(b) Determine a model that best fit the data based on the adjusted R^2

.

(c) Write down the equation of the best fitted curve that you have selected from part (b)

Sol:

(a)

Polynomial model	Adjusted R^2	AIC	BIC
First order	0.861	431.5	433.0
Second order	0.996	374.6	376.9
Third order	1.0	289.8	292.9
Fourth order	1.0	5.209	9.072

(b) Since the fourth-order polynomial has the largest adjusted R^2 thus, model with polynomial fourth degree should be selected.

(c) $y = 12.3128 + -2.55x + 0.4588x^2 + 0.4921x^3 + 1.1201x^4$

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
y = np.array([1,191,032,183,297,183,297,96,071,699,577,11,726,1,346,458,1
              445,462,267,807,1,049,467,24,124,222,556,222,556,804,549,96
print(y)
x = np.array([32,20,20,17,28,10,33,19,
              25,22,31,12,21,21,29,17]]).reshape((-1,1))
x1st = PolynomialFeatures(degree=1, include_bias=True).fit_transform(x)
x2nd = PolynomialFeatures(degree=2, include_bias=True).fit_transform(x)
x3rd = PolynomialFeatures(degree=3, include_bias=True).fit_transform(x)
x4th = PolynomialFeatures(degree=4, include_bias=True).fit_transform(x)
model1 = sm.OLS(y, x1st)
results1 = model1.fit()
model2 = sm.OLS(y, x2nd)
```

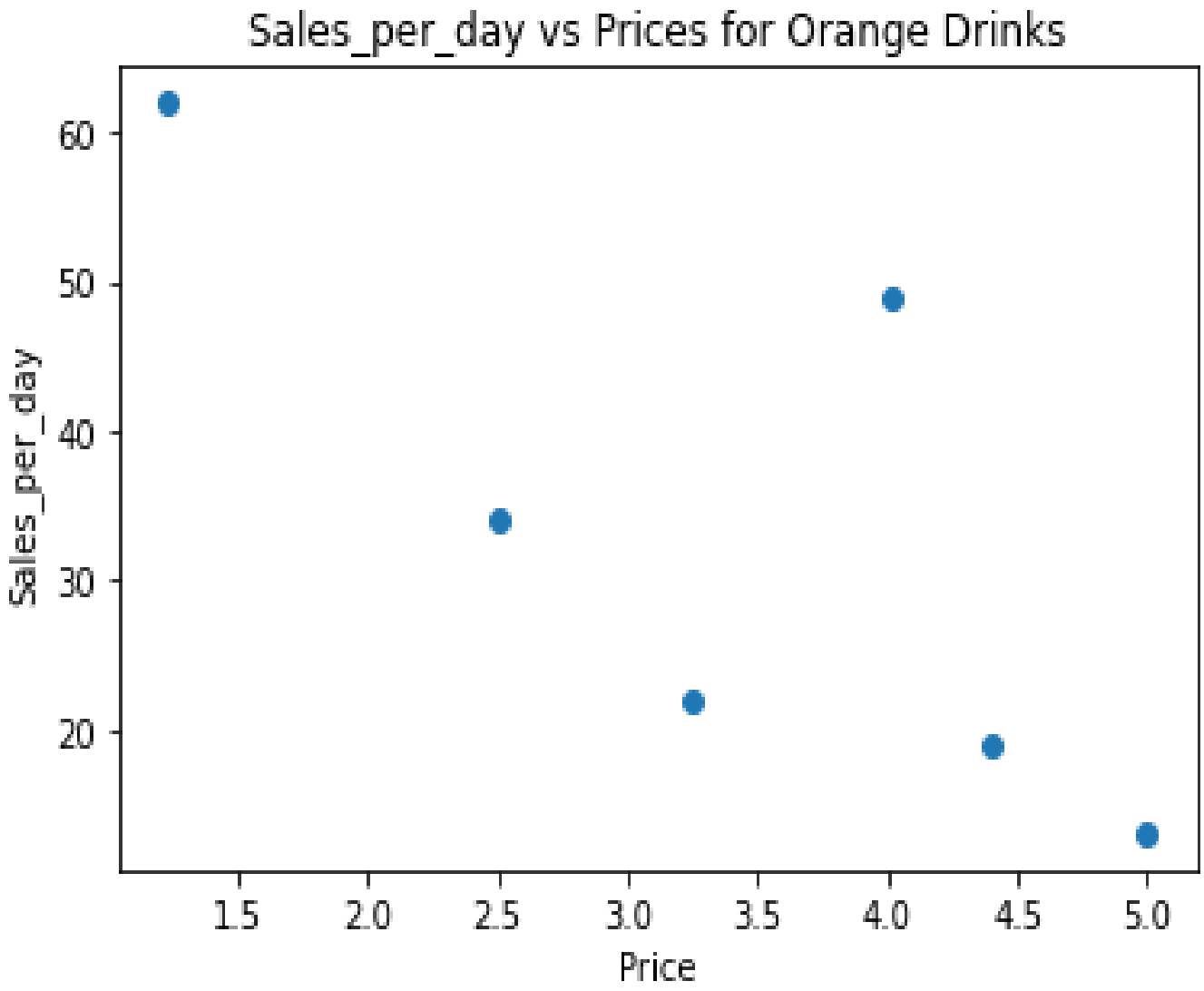
```
results2 = model2.fit()
model3 = sm.OLS(y, x3rd)
results3 = model3.fit()
model4 = sm.OLS(y, x4th)
results4 = model4.fit()
print("Adj. R-squared model1 = ", results1.rsquared_adj)
print("AIC model1 = ", results1.aic)
print("BIC model1 = ", results1.bic)
print("Adj. R-squared model2 = ", results2.rsquared_adj)
print("AIC model2 = ", results2.aic)
print("BIC model2 = ", results2.bic)
print("Adj. R-squared model3 = ", results3.rsquared_adj)
print("AIC model3 = ", results3.aic)
print("BIC model3 = ", results3.bic)
print("Adj. R-squared model4 = ", results4.rsquared_adj)
print("AIC model4 = ", results4.aic)
print("BIC model4 = ", results4.bic)
print("Beta hat = ", results4.params)

Output:
Adj. R-squared model1 =  0.861
AIC model1 =  431.5
BIC model1 =  433.0
Adj. R-squared model2 =  0.996
AIC model2 =  374.6
BIC model2 =  376.9
Adj. R-squared model3 =  1.0
AIC model3 =  289.8
BIC model3 =  292.9
Adj. R-squared model4 =  1.0
AIC model4 =  5.209
BIC model4 =  9.072
Beta hat =  [12.3128 -2.5477 0.4588 0.4921 1.1201]
```

6.2 Visualizing Data in Python

A scatter plot is a visual representation of how two variables relate to each other. You can use scatter plots to explore the relationship between two variables, for example by looking for any correlation between them.
One of the most popular modules is **Matplotlib** and its submodule **pyplot**, often referred to using the alias **plt**. Matplotlib provides a very versatile tool called **plt.scatter()** that allows you to create both basic and more complex scatter plots.
A cafe sells six different types of bottled orange drinks. The owner wants to understand the relationship between the price of the drinks and how many of each one he sells, so he keeps track of how many of each drink he sells every day. You can visualize this relationship as follows:

```
import matplotlib.pyplot as plt
price = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
sales_per_day = [34, 62, 49, 22, 13, 19]
plt.scatter(price, sales_per_day)
plt.title("Sales_per_day vs Prices for Orange Drinks")
plt.xlabel("Price")
plt.ylabel("Sales_per_day")
plt.show()
```


Sales_per_day vs Prices for Orange Drinks
Sales_per_day
60
50
40
30
20
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Price

Example 4.

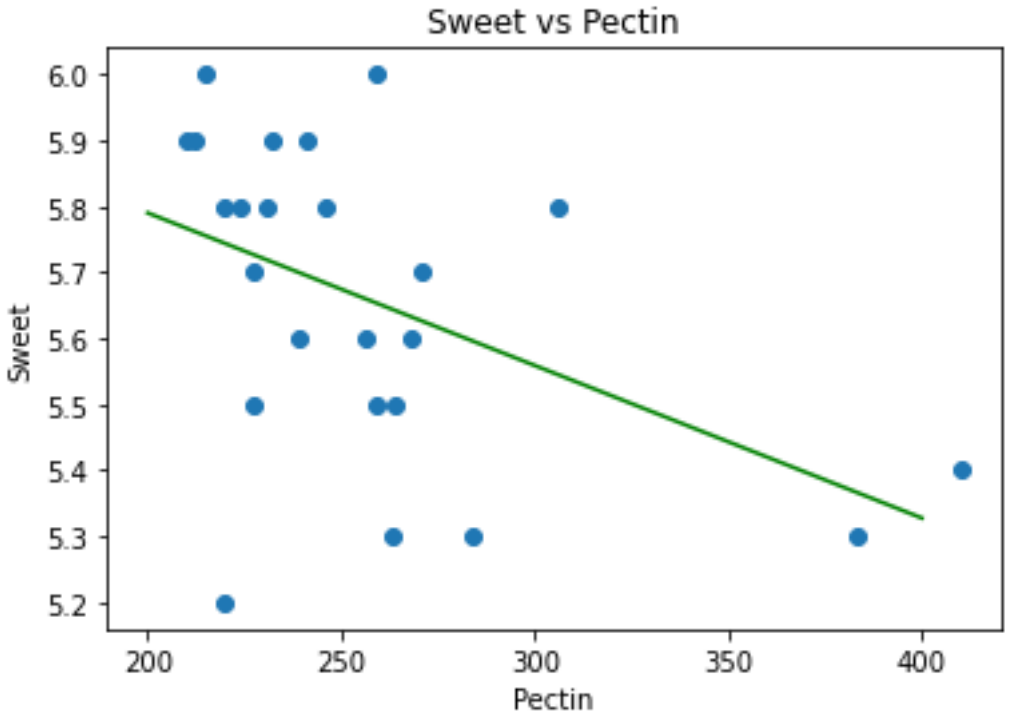
The quality of orange juice produced by a manufacturer is constantly monitored. Is there a relationship between the sweetness index and chemical measure such as the amount of water soluble pectin (parts per million) in the orange juice? Data collected on these two variables for 24 production runs at juice manufacturing plant are save in orjuice.csv file. Suppose a manufacturer wants to use simple linear regression to predict the sweetness(Y) from the amount of pectin (X).

(a) Fit the data to the model $Y = \beta_0 + \beta_1 X + \epsilon$.

(b) Plot the fitted regression function and the data.

Sol:

(a) $\hat{y} = 6.252067909048171 - 0.00231062588246409x$



(b)

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
df = pd.read_csv('orjuice.csv')
#print(df)
model1 = np.poly1d(np.polyfit(df.Pectin, df.Sweet, 1))
print(model1)
plt.scatter(df.Pectin, df.Sweet)
plt.title("Sweet vs Pectin")
plt.xlabel("Pectin")
plt.ylabel("Sweet")
polyline = np.linspace(200, 400, 500)
plt.plot(polyline, model1(polyline), color='green')
```

Example 5.

You are given the following data:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
y	3	14	23	25	23	15	9	5	9	13	17	24	32	36	46

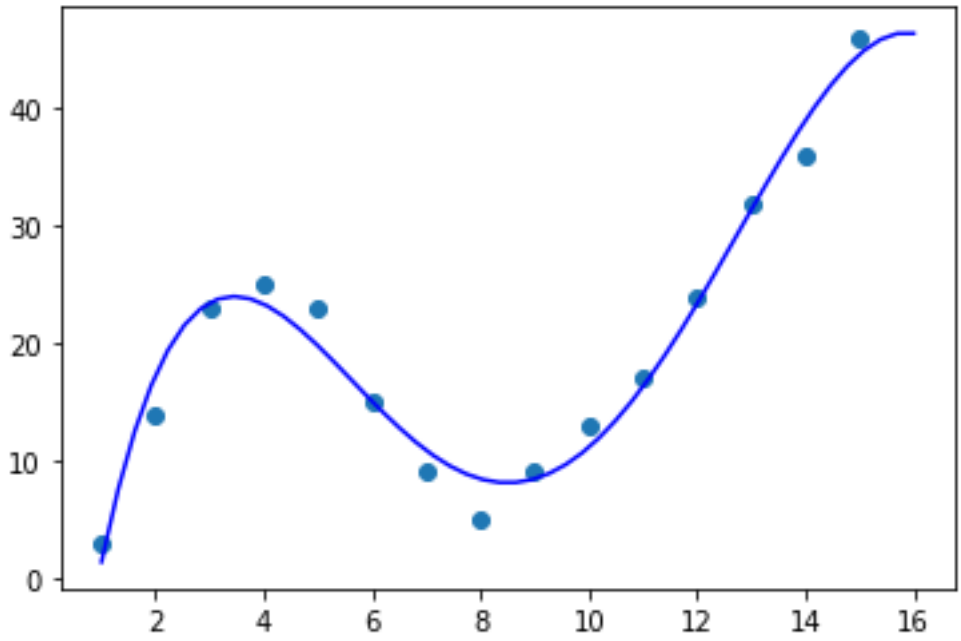
Fit the data to the polynomial regression of degree 5. Created a scatter plot of the data and add the fitted polynomial lines to scatterplot.

Sol:

```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
y = [3, 14, 23, 25, 23, 15, 9, 5, 9, 13, 17, 24, 32, 36, 46]
#fit polynomial models of degree 5
model5 = np.poly1d(np.polyfit(x, y, 5))

#create scatterplot
polyline = np.linspace(1, 16, 50)
plt.scatter(x, y)

#add fitted polynomial lines to scatterplot
plt.plot(polyline, model5(polyline), color='blue')
plt.show()
```



6.2.1 Analysis of Variance(ANOVA)

The f_oneway() function of the SciPy library can perform ANOVAs using one-way methods.

Example 6.

A researcher took 20 cars of the same to take part in a study. These cars are randomly doped with one of the four-engine oils and allowed to run freely for 100 kilometers each. At the end of the journey, the performance of each of the cars is noted. The results follow.

Engine Oil	Performance				
1	89	89	88	78	79
2	93	92	94	89	88
3	89	88	89	93	90
4	81	78	81	92	82

Consider the model $y_{ij} = \mu_i + \epsilon_{ij}, i = 1, 2, 3, 4; j = 1, 2, \ldots, 5$, where

- y_{ij} is the performance of the j^{th} car with i^{th} engine oil.
- μ_i is the mean performance of car with i engine oil.
- $\epsilon_{ij} \sim N(0, \sigma^2)$

The hypotheses to test the equalities of means are H_0 : $\mu_1 = \mu_2 = \mu_3 = \mu_4$ versus H_1: at least one population mean is different from the rest. Use the f_oneway package to compute the test statistic and the p value.

Sol:

F = 4.625000000000002 p value = 0.016336459839780215

```
from scipy.stats import f_oneway
y1 = [89,89,88,78, 79]
y2 = [93,92,94,89, 88]
y3 = [89,88,89,93, 90]
y4 = [81,78,81,92, 82]
ANV = f_oneway(y1,y2,y3,y4)
print("ANOVA = ", ANV)
print("F = ", ANV[0])
print("p value = ", ANV[1])
Output:
ANOVA = F_onewayResult(statistic=4.625000000000002,
F = 4.625000000000002
p value = 0.016336459839780215
```